Molecular Dynamics: Lennard-Jones liquid

Florian Dommert, Nadezhda Gribova, Peter Košovan*

17th November 2010

This tutorial aims on providing a first touch with **Molecular Dynamics (MD)** simulations. In the previous tutorial we learned about some integrator algorithms and followed trajectories of planets in the solar system. In this tutorial, we will simulate a collection of particles interacting via the Lennard-Jones potential. The focus of the tutorial is to follow how the system equilibrates. In the first example, we will show that a system such as a dilute gas or a liquid reaches within several simulation steps a state with the same statistical properties, irrespective of the initial configuration. In the second example we show that in some systems such as a crystal or a glass this may not happen within an acceptable simulation time.

1 Introduction

In the previous tutorial we were simulating the solar system and were examining the trajectories of planets. In this tutorial, we will simulate a much denser system of soft spherical particles. Contrary to the planetary system, this one is dominated by collisions and therefore individual particle trajectories strongly depend on initial conditions. In such a system, we do not study individual trajectories anymore, but rather their statistical properties which are related to macroscopic observables.

2 Implementation of the Force Field

In the previous tutorial we were implementing various integrators including velocity verlet which has been implemented in the code provided to you now. An inevitable input for the integrator are forces. In this tutorial you will be expected to implement the calculation of forces.

We are going to simulate a system consisting of Lennard-Jones particles, i. e. particles interacting with the Lennard-Jones (LJ) potential. This potential has been originally derived for a system of noble gas atoms, where the interactions between induced dipoles in the atoms dominate on longer distances. It can be shown that this interaction has an r^{-6} dependence, which is the second term in Equation 1. Apart from the attractive interaction, there is also a short-ranged repulsive contribution, about the exact r-dependence of which we have much less information apart from the fact that it is much steeper than the attractive part. Hence, for computational convenience, r^{-12} has been used for the repulsive part, leading to the final potential:

$$V_{\mathrm{LJ}}(r) = 4\epsilon\left(\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right) . \tag{1}$$

If we wanted to simulate the full LJ potential, we would quickly run into problems, since its interaction range is infinite and we can only simulate a finite system. Fortunately, it converges fast enough to zero and we can introduce an approximation by truncating it at a certain distance and adding an appropriate shift so that it is continuous:

$$V_{\mathrm{LJ}}(r) = \begin{cases} V_{\mathrm{LJ}}(r) - V_{\mathrm{LJ}}(r_{\mathrm{cutoff}}) & r \leq r_{\mathrm{cutoff}} \\ 0 & r > r_{\mathrm{cutoff}} \end{cases} . \tag{2}$$

*kosovan@icp.uni-stuttgart.de

For $r < r_{\mathrm{cutoff}}$ the forces due to the truncated LJ potential are the same as due to the full form. In practice, $r_{\mathrm{cutoff}} = 2.5\,\sigma$ is often used. To obtain thermodynamic properties of a system interacting with the full LJ potential from a simulation performed using the truncated version, a correction has to be added. If the cutoff radius r_{cutoff} is chosen large enough, a correction to the total energy can be estimated:

$$E_{\mathrm{corr}} = \int_0^\infty \mathrm{d}r V_{\mathrm{LJ}}(r_{\mathrm{cutoff}}) g(r)\,, \tag{3}$$

where $g(r)$ (see also further below) obtained from the simulation is used. Because $g(r \to \infty) = 1$, the integral converges only for potentials decaying faster than r^{-3}. For interaction potentials decaying slower than r^{-3}, special methods have to be used.

2.1 Reduced units

For simple liquids composed of particles which interact with the same pairwise-additive potential, one can write the potential in a general form:

$$U(r) = \epsilon\phi(\sigma/r) \tag{4}$$

Then it is possible to define a set of dimensionless reduced units such as $x^* = x/\sigma$, $V^* = V/\sigma^3$, $T^* = k_{\mathrm{B}}T/\epsilon$ and $P^* = P\sigma^3/\epsilon$. Within this set of reduced units, all systems interacting with potentials of the form given by equation (4) follow one single universal equation of state. Therefore, results of simulations of simple fluids are presented in reduced units. The results can then be transferred to a particular temperature and pressure, when values of ϵ and σ for a particular fluid are known. One particular implication of the theorem of corresponding states is that we can perform all simulations with $\epsilon = 1.0$ and $\sigma = 1.0$ and if desired, the results can be transfered to other values of interaction parameters.

2.2 Homework (2 points)

1. Modify the file `force.c` to enable the calculation of the interaction energies and forces due to the truncated LJ interaction potential with with $\epsilon = 1.0$, $\sigma = 1.0$ and $r_{\mathrm{cutoff}} = 2.5$. *Pay attention: So far we have just talked about potentials and not forces!* All necessary variables have been declared for you, only insertion of some lines is required. After you have inserted all the code type `make` to produce an executable called `md_nve`.

2. In the implementation, first the value of $|\vec{r}||\vec{F}|$, is computed and stored in variable `Ff`, then it is divided by r^2 and later on multiplied by the components of $\vec{r}$: (x, y, z). This is definitely not the most straightforward way. Try to explain why one should prefer this implementation rather than directly computing $|\vec{F}|$ and multiplying it by components of a unit vector in the direction of $\vec{r}$: $(x/r,\, y/r,\, z/r)$.

3 Running the simulation

3.1 Input of parameters

The input parameters for the simulation program are provided via the input file

```
input.in
```

This is one of the usual ways of providing input to simulation programs. Notice that this file requires the input parameters in a fixed order and only little checking of the sanity of the input parameters is performed.

The output is written into files `observables.dat` (Total, Potential and Kinetic energy, Temperature, Pressure), `rdf.dat` (radial distribution function), `VelDist.dat` (velocity distribution) `vacf.dat` (velocity autocorrelation function) and `msd.dat` (mean-square displacement). The last two observables we will not discuss in the tutorial but if you are interested, you may examine them on your own. If files with the same names happen to be already present, they will be overwritten. When performing a series of simulations at different conditions, you have to rename output files or move them to somewhere else. It is a good practice to use file names which tell you the parameters at which the simulation has been performed.

If you have implemented the forces and your code compiles without an error, run a simulation and plot the total energy from the output file `observables.dat`. If the total energy is not conserved, search for a mistake

in your code. If you do not switch off the velocity rescaling by setting the number of rescaling steps to zero, energy will not be conserved while rescaling is performed (see also further below). On the other hand, if you switch off the velocity rescaling, you may need to switch on the pre-equilibration to prevent the system from blowing up due to very unfavourable initial configuration.

3.2 Initial system setup

The first step when starting a simulation is to set up the system. In the code provided to you, there are three options of initial system configuration:

1. completely random setup
2. simple cubic lattice
3. hexagonal close-packed lattice

In a molecular simulation we are not interested in the exact trajectories of individual particles but rather in their stochastic properties which are given by the macroscopic variables such as temperature and density. Therefore the observables computed from such a simulation should not depend on initial conditions. We will test this by performing simulations at the same temperature and pressure but with different symmetry of the initial configuration.

3.3 Equilibration

Usually we do not know at the beginning of the simulation, what the system looks like under the simulation conditions. On the contrary, we are often performing the simulation to be able to answer this question. Due to the very high number of possible system configurations, it is very unlikely that our initial configuration is a typical one for the system in equilibrium. Fortunately, due to the chaotic nature of the many-body system, irrespectively of the initial configuration it finally reaches the same state corresponding to given macroscopic observables.

The time-evolution of the system from the initial configuration towards one which corresponds to equilibrium is called *equilibration*. It is manifested by a time-drift of all observables. On the contrary, once the equilibrium state is reached, *all* observables fluctuate around a constant value. We would like to point out that while some observables relax to their equilibrium values very quickly, others may take much longer. How long the equilibration takes depends on system properties as well as on how far from equilibrium the initial state is. A system cannot be considered as equilibrated is there exists one single observable which exhibits a time-drift.

Unfortunately, there is no universal recipe, according to which one can safely tell that the system has equilibrated, i. e. the equilibration period has finished and one can start collecting data. When the measured quantities do not drift anymore, most probably the system is equilibrated but this is no guarantee! To decide if we can start collecting data, we have to make sure that the time over which the observables are fluctuating around a constant value exceeds many times the duration of equilibration, but also we may need to apply some physical knowledge of the simulated system.

When we know that the starting configuration is very far from equilibrium, it may be useful to use some other method to bring it quickly closer to equlibrium before the simulation is started. One such method, which is implemented in our code, is called *steepest descent energy minimization*. It is switched on and off by setting the appropriate variable in the input file to 1 or 0.

3.4 Data collection and measurement of observables

After the system has equilibrated, we can start measuring observables which will then be used for the computation of ensemble averages. To measure observables in MD, we make the use of the ergodic hypothesis which states that an ensemble-average of an observable is equal to its time-average over a long-enough time interval:

$$\langle A \rangle = \lim_{t_{\max} \to \infty} \frac{1}{t_{\max}} \int_0^{t_{\max}} A(t)\, \mathrm{d}t \tag{5}$$

In other words, if we follow the evolution of the system long enough, it visits each point of the configurational space with the proper probability corresponding to the simulated statistical ensemble. If we measure instantaneous values of A at regular intervals Δt then the ensemble average is computed from simulation as

$$\langle A \rangle = \frac{1}{N} \sum_{i=0}^{N} A(i\Delta t) \tag{6}$$

When computing ensemble averages, data from the equilibration have to be discarded. In the current code, we measure the observables on the fly while the simulation is running and compute the ensemble average at the end. The variable `Nsteps_equil` tells the program after how many time steps it starts collecting data to compute the averages. When starting the simulation, we do not know yet how long the equilibration will take. We can only make an educated guess. Therefore it is always necessary to check at the end, that the equilibration has been shorter than `Nsteps_equil` we set at the beginning and if not, we have to rerun the simulation again with a higher value of `Nsteps_equil`.

In this tutorial we will be measuring several observables. While measuring the total energy of the system, the kinetic energy and the potential energy is trivial, computation of other observables requires some knowledge of statistical mechanics. We will introduce some of them here. To measure the temperature, we make the use of equipartition theorem from statistical mechanics which states that kinetic energy per degree of freedom is $1/2\, k_{\mathrm{B}}T$. Point-like particles have 3 translational degrees of freedom each, and hence the temperature is computed as follows:

$$T = \frac{2}{3} \frac{U_{\mathrm{kinetic}}}{N k_{\mathrm{B}}} \tag{7}$$

From statistical mechanics it can be shown that Pressure can be computed from two contributions. One of them is the ideal-gas contribution, the other one comes from the interactions. Without giving details, we just state the formula:

$$P = \frac{1}{V} \left(\frac{1}{3} \sum_{i=0}^{N} m v_i^2 + \sum_{i=0,\, j<i}^{N} \vec{F}_{ij} \cdot \vec{r}_{ij} \right) \tag{8}$$

An observable containing information about the structure of the system is the of the radial distribution function (RDF) defined as

$$g(r) = \frac{1}{\rho 4\pi r^2 \mathrm{d}r} \sum_{ij} \langle \delta(r - |r_{ij}|) \rangle \tag{9}$$

The RDF describes by how many particles a particle is surrounded at a particular distance in comparison with the ideal gas phase, where we have $g(r) = 1$ at every distance r. It can be used to compute the equation of state and all kinds of thermodynamic observables. For our purpose we mention three typical shapes of $g(r)$. In a dilute gas, it is a monotonously decaying function. In a simple liquid it has a form of oscillations which are damped out with increasing r. In an ideal crystal, $g(r)$ contains sharp peaks at well-defined positions which are given by the symmetry of the lattice.

3.5 Simulating at a desired temperature – velocity rescaling

Because we are using the energy-conserving velocity verlet algorithm and we keep a constant number particles in a box of constant volume, by construction we are simulating the microcanonical $[N, V, E]$ - ensemble. However, it is close to impossible to perform experiments at constant total energy of a system. On the other hand, experiments at constant temperature and density are routinely performed. To be able to compare simulations to experimental data, one would prefer to simulate at a pre-defined temperature, rather than at total energy, i. e. to simulate a canonical $[N, V, T]$ - ensemble. One way of driving the simulated system to the desired temperature is rescaling particle velocities. In this procedure, velocities of all particles are multiplied by a factor such as to achieve the desired temperature (equation 7). This is repeated until the temperature remains constant. The number of steps of velocity rescaling in our program is controlled by the `Nsteps_rescale` variable.

One important problem of the velocity rescaling is that it destroys the Maxwell-Boltzmann distribution of velocities in the system. Therefore, a simulation with velocity rescaling produces neither $[N, V, E]$ - nor $[N, V, T]$ - ensemble. In the implementation here, the velocity rescaling is only used in equilibration to drive the system to the desired temperature. During the productive simulation run, no velocity rescaling is performed anymore,

hence we are simulating an $[N, V, E]$ - ensemble with (average) temperature close to our pre-defined value. In the next tutorial, you will learn about more advanced methods of simulating at a constant temperature.

3.6 Homework (1 point)

1. Explain why it is interesting to rescale the velocities, if we want to obtain a constant temperature.

2. Find which lines in file `mdloop.c` are responsible for the velocity scaling. *Just provide the line numbers.*

4 Study and analysis of the results

The main point of this tutorial is to perform a series of simulations, to follow the time-evolution of various observables and to identify the equilibration period (if the system equilibrates at all).

4.1 Homework (3 points)

In this part of the homework, you will follow an equilibration of a system with a given initial structure and a uniform initial velocity distribution.

1. Perform MD simulation at density $\rho = 0.8$ with hexagonal close-packed (hcp) configuration as a starting point and velocity rescaling switched *off* (set `Nsteps_rescale` to zero). Set temperature to 1.0. This will be used to set the initial velocity distribution but the final temperature will be different. For reference, provide the input file of the simulation.

2. Plot the time-evolution of total energy to show that it is conserved up to computer precision.

3. Plot time-evolution of all other observables to illustrate how they relax to constant values. Becareful to zoom the plots appropriately on the y-axis.

4. Identify an observable which relaxes most slowly. By looking at the time-evolution of the most slowly relaxing variable, estimate the duration of equilibration. Make sure that it is only a fraction of the whole simulation. You may need to perform a longer simulation to achieve that. *Note that this has nothing to do with the `Nsteps_equil` variable you set in the input.*

5. Plot the velocity distribution and fit it with the Maxwell-Boltzmann distribution function. Provide the velocity obtained from the fit and compare it to the one computed as an ensemble average.

4.2 Homework (2 points)

In this part of the homework, we will use velocity rescaling during equilibration to drive the system to a desired temperature. We will also examine the effect of initial structure on the final properties of the system.

1. Simulate the same system as you did in the previous case at $T = 1.0$ and switch on the rescaling taking `Nsteps_rescale` $\leq$ `Nsteps_equil`. If your final temperature deviates from the desired one by more than 10%, make the rescaling period longer. Notice that after the end of velocity rescaling the kinetic energy may take some more time to relax.

2. Perform two more simulations of the same system with different symmetry of the initial configuration.

3. Plot the time evolution of pressure in all three systems (in separate plots) to show that they differ.

4. In a table, compare the average values of all observables for all three different systems. Are they identical within the estimated error?

5. Plot the radial distribution function of all three systems in one plot to show that they have identical structure.

4.3 Homework (2 points)

In the last part of the homework, we will use the same approach as in the previous but we will focus on cases when it may take longer for the systems to equilibrate. You may try to switch on the pre-equilibration to speed up the equilibration.

1. Take your system from the previous example and decrease the density to $\rho = 0.2$.
2. Repeat the simulation with different initial conditions and plot the RDF to see if the structures differ. If you increase the number of simulation steps, will the structures be closer to each other?
3. Take the same system again and increase the density to $\rho = 1.2$.
4. Repeat the simulation with different initial conditions and plot RDF to see if the structures differ. When you increase the number of simulation steps, will the structures be closer to each other?

4.4 Optional homework (2 points)

1. Explain, why equilibration takes longer in some cases. In different cases, the reasons may differ!
2. Explain, why in some cases the structures may differ even though both systems seem to be equilibrated and no further drift of observable values can be seen.

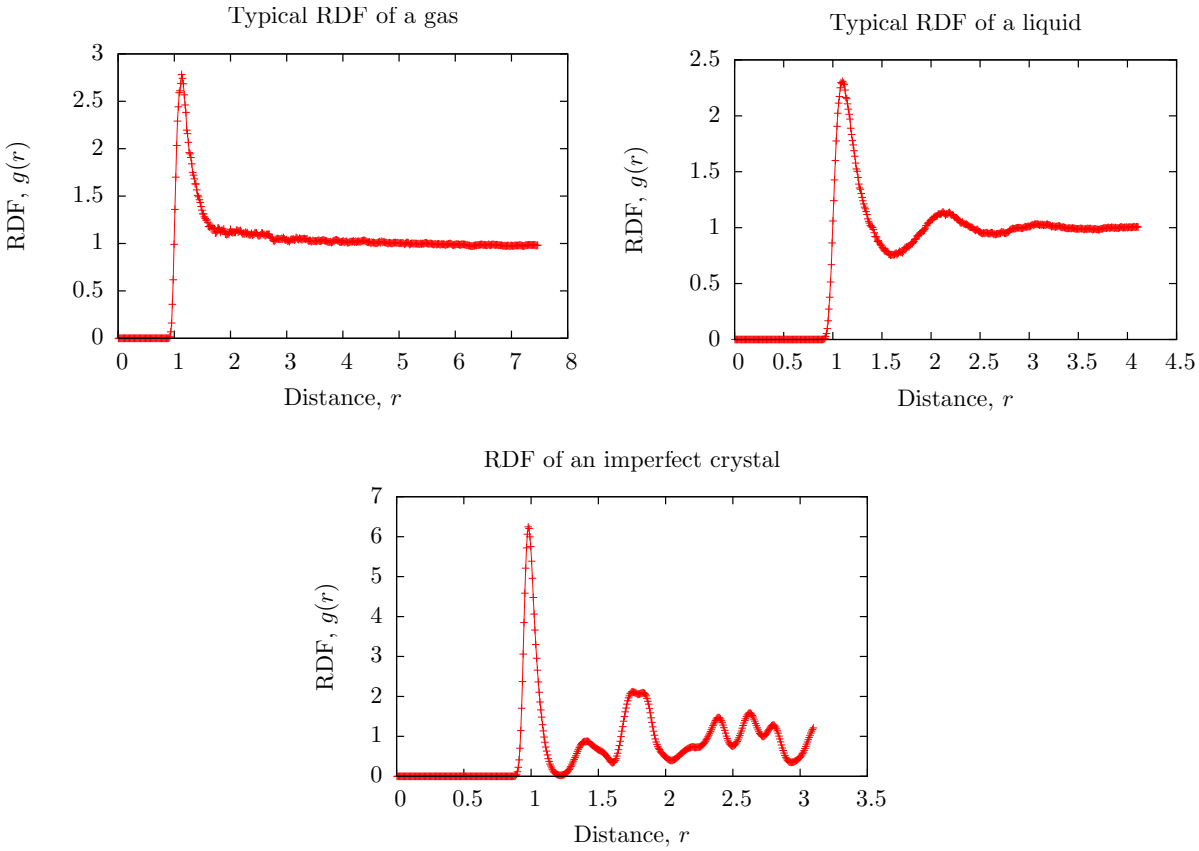

Figure 1: Examples of radial distribution functions obtained from simulations.